ESD-TR-69-384

Semiannual Technical Summary
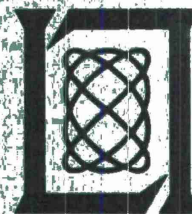
Graphics

30 November 1969

Prepared for the Advanced Research Projects Agency
under Electronic Systems Division Contract AF 19(628)-5167 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Lexington, Massachusetts

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

GRAPHICS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
ADVANCED RESEARCH PROJECTS AGENCY

1 JUNE – 30 NOVEMBER 1969

ISSUED 7 JANUARY 1970

LEXINGTON   MASSACHUSETTS

SUMMARY

Graphical output and interactive input routines have been written for BCPL, offering a useful alternative to assembly language for writing graphical subsystems on TX-2. Design work is under way on the system architecture and software for a new terminal support system intended to serve as many as 20 interactive but nondynamic graphic consoles. A storage scope editor has been implemented on TX-2 with the intent of exploring some of the problems to be encountered in the new system which will have storage scopes for display output.

Experiments with the color display on TX-2 await the delivery of a new CRT with longer persistence. A box, or rectangle, generator was designed and installed in an attempt to reduce the display flicker for the semiconductor mask design application. The resulting improvement prompted detailed measurement of display system performance. A new character generator based on the stroke writing principle has been built and is being checked out.

A program written to demonstrate the application of interactive graphics to regional planning has incidentally shown that the storage scope can provide quite adequate eight-level gray-scale area maps. Conclusions drawn from two years' experience with programs that aid in the design of semiconductor masks have led to the design of a radically different, high performance, new system for integrated circuit layout and mask making. Progress has been made in the development and validation of a high-speed algorithm for testing the planarity of a graph which is expected to have application in the layout problem.

CONTENTS

GLOSSARY

ALGOL	A high-level algebraic problem-solving language
BCPL	Basic Combined Programming Language — an intermediate level language for computer programming
LIL	Local Interaction Language — a high-level language for use in the TSP system
LX-1	A prototype microprocessor being constructed at Lincoln Laboratory
LEAP	Language for Expressing Associative Procedures — an ALGOL-like TX-2 programming language
MK 5	A TX-2 assembly language
TSP	Terminal Support Processor

GRAPHICS

I. LANGUAGES AND SYSTEMS

A. Graphics in BCPL

The LEAP routines which allow graphical output and interactive input have been rewritten for BCPL.[†] Their names and usage have been kept as compatible as possible with LEAP, with the following exceptions:

(1) Scope coordinates in BCPL are integers with values in the range ±1000 in both X and Y.

(2) No conics capability has been implemented.

With these graphical tools, BCPL is now a useful alternative to assembly language for such subsystems as graphical debuggers and graphical text editors. Even though assembly language for these applications is somewhat more efficient, the added advantages of relocatable code, ease of modification and ease of transfer of responsibility from one programmer to another, make BCPL an attractive alternative.

In other situations a programmer now has a choice between BCPL with graphics and LEAP. For applications where the data structuring is relatively simple, it is far better to use BCPL since, in LEAP, the considerable overhead of the associative language is invoked and yet its power is hardly used at all. As an experiment, a simple program was written in both BCPL and LEAP; it allows a user to enter and edit poles and zeros with a tablet and then request the magnitude of the frequency response to be displayed. Timing and space measurements showed the BCPL-coded program to run twice as fast using only one-third as much space.

For large complicated programs such as the semiconductor mask program, it would be unfeasible to use BCPL because producing the data structure necessary would be very painful, and also debugging would be very difficult, since there is no run-time type checking.

In summary, BCPL with graphics covers the middle ground between Mk 5 and LEAP where more efficiency than LEAP and the advantages of a higher-level language are desired.

B. Terminal Support System

Design work is under way on the system architecture and software for a new terminal support system. The design assumes an independent processor configuration called TSP (for Terminal Support Processor) serving as many as 20 consoles. Each console is to consist of a keyboard, tablet, and a pair of storage scopes. The scopes are to be driven from a common display generator (vectors and characters). The TSP is to be connected to both TX-2 and the IBM 360/67, and the consoles it supports can act as user consoles for either or both of those large time-sharing systems. Communications between the TSP and the large machines are to be compatible with ARPA network conventions insofar as possible.

[†]Graphics, Semiannual Technical Summary to the Advanced Research Projects Agency, Lincoln Laboratory, M.I.T. (30 November 1968), DDC AD-679991, p. 1.

Support services to be provided by the TSP system are:

(1) Input

Keyboard echoing and buffering

Tablet tracking and inking

Character recognition

(2) Output

Maintenance of structured display files

Scaling and windowing on output to the display generator

Interactions between console input and output, and between TSP and the large computers, are to be under user program control. A high level language to be called LIL (for Local Interaction Language) is being designed for the purpose of specifying the output displays as well as controlling the interactions. As the design is now developing, LIL statements will be compiled in TX-2 or the 360/67 to a form analogous to assembly code which will be interpreted by the LX-1 microprocessor, a part of the TSP processor configuration.

On the basis of measurements of the TX-2 graphic support system, estimates of the processor and memory requirements for the TSP have been made, and the documents necessary for the procurement of the other components of the system are being drafted.

Software design effort is concentrated on the specification of the LIL language, particularly the techniques it will use in handling the virtual memory in which the display and interactive programs will operate.

C. Storage Scope Editor

The design and implementation of a text editor for the Tektronix Type 611 storage scopes on TX-2 have been undertaken with the intent of exploring some of the problems to be encountered in the design of the new terminal support system. Since it could be expected to reduce the loading of the display generator by roughly two orders of magnitude, general acceptance of a storage scope editor by TX-2 users would greatly improve the display flicker situation for other users of the refresh displays. To date, a successful (but by no means final) version has been implemented and has attracted a few users.

A feature of the Tektronix Type 611 storage scope is a mode called "write-thru" which allows some information to be displayed in a nonstoring mode without disturbing other stored information present on the screen at the same time. Experiments show that the successful use of write-thru mode in a lighted room requires careful control (and uniformity) of writing speed. Information displayed in that mode is not easily seen because the intensity is very low, and useful results are not obtained except in the border line case where some storage is taking place. This consideration led to the restriction of the use of write-thru mode in the editor to the display of cursors. Experience shows that users will tolerate partial storing of cursors in return for greater visibility.

A further consideration regarding the restriction of write-thru to cursor display was the problem of system loading. In the TSP system a single display generator will have to service about 20 scopes. It has been determined that write-thru is most visible if the refresh rate is about 4 to 6 times per second. For this rate, full loading of the display generator would occur with 10 msec of display for each user. Of course, full loading is not acceptable, since it would not allow for generating the stored display. Drawing a cursor presently takes nearly 1 msec.

Since the editor calls for two cursors (one for the text and one for echo of typed characters), generator loading for 20 scopes would be about 20 percent, which seems acceptable.

A basic design requirement for the editor is to minimize the need to update the display, since any change other than a simple addition will require an erase and complete re-painting. An erase cycle is undesirable from both the system and the user standpoints. To the system, the repainting loads the display generator; to the user, the erase cycle inevitably causes some disorientation. This requirement was met in the editor by using rather powerful commands and conventions which avoid regenerating the display until clearly necessary.

Since response to a command is not likely to be immediate, it is desirable that many commands should be processed without a response being required. No response is given until a carriage return is encountered, and then only if another command is not ready. Any number of commands may be specified on each line, so that the user can control the rate of regeneration.

In the interests of quick typing and limited complexity of the editor, it was decided that all commands should be single alphabetic characters and should fall into one of three groups according to the number and kind of parameter expected. To further reduce the typing required of the user, careful attention was given to default conditions for missing parameters.

The editor commands are as follows:

> T — Move to top of file
> B — Move to bottom of file
> P — Move forward a page
> p — Move backward a page
> L — Locate line containing string
> N — Move cursor within line
> ⟨Integer⟩ — Move cursor that many lines

These commands move the cursor within the file. The display is not regenerated unless the cursor no longer falls within the displayed text, in which case it is regenerated with the cursor on the top line.

> D — Delete lines
> X — Delete characters
> K — Insert string
> I — Insert line (string plus carriage return)
> C — Change string

These commands change the text, so the display is always regenerated after them. If the display is regenerated because of one of these commands, the display will start the same number of lines before the cursor as it did before the command was given (to avoid disorientation).

> F — Specify file
> G — Get lines from input pointer
> M — Move lines from input pointer
> V — View input source
> W — Return to working file

These commands give the facility of bulk text input from external files. Implementation is not completed.

TABLE I

SUMMARY OF TX-2 DISPLAY SYSTEM MEASUREMENT RESULTS

| | Display Frame Time | = CPU Time to Service + Display | Display System Busy Time | Display System Busy Time = | | | | No. of Data Words |
				Line Positioning + Time	Line Drawing + Time	Character Positioning and + Writing Time	Display System Overhead†	
Mask 1	54.9	9.7	45.2	1.8	16.9	18.4	8.1	1027
Mask 2	54.1	8.8	45.3	1.7	18.2	18.4	7.0	877
Mask 3	77.4	21.5	55.9	2.8	24	18.4	10.7	1553
Mask 3 Using Vector Mode	142	29.5	112.5	6.9	66.9	18.4	30.3	2944

† Display overhead consists of adder propagation time, deflection amplifier feedback circuit delay time and other logic delay time.

All times measured in milliseconds.

Number of characters displayed is approximately 160.

Δ — Dump as text file

E — Exit with changes

α — Abort (exit without changes)

These commands interface the working file to the outside world.

β — Execute BT command

This command allows the user to run any public or private program using the basic translator (BT) calling conventions.

User feedback indicates this form of editor is powerful enough to handle the standard situations. However, it will probably be necessary to add facilities for bulk keyboard input and iteration for some commands.

Complaints include difficulty in distinguishing the cursor and sluggish response. It is not clear that anything can be done about the cursor, but sluggish response can probably be somewhat relieved by careful paging of the text, keeping text near the cursor frozen in core. The program itself is small enough to be kept frozen in core.

Perhaps the most important result of this effort has been that it has forced us to re-think our ideas on structured displays. The standard TX-2 display structure has proven very poorly suited to the requirement of a storage scope editor, and we have had to design a user-structured display file for storage scopes. This contains a list of display items, with flags on each item to indicate if it has been stored, pointers available to both system and user, and a rudimentary subroutine facility.

D. Display Hardware

1. Color Display

Experiments with a two-layer red-green phosphor in a standard TX-2 display tube have been suspended pending the delivery of a tube with a longer persistence phosphor. The available tube has a TV type (P22) phosphor which requires too high a refresh rate to be usable in the TX-2 system. The utility of a two-color display cannot be realistically evaluated until the persistence problem is overcome.

2. Display Measurements

In an attempt to reduce the display flicker for the semiconductor mask design application, a box, or rectangle, generator was designed and installed in the TX-2 display system. Since the mask design program causes large numbers of rectangles to be displayed, the use of the box generator was expected to reduce memory requirements substantially and to increase drawing speed by a significant factor. The improvement actually realized was more modest than had been expected. Memory requirements were reduced as anticipated, but the display frame time was decreased by only a factor of two. Since a larger factor had been expected, a series of measurements of display system performance were undertaken using the mask design program to provide test cases. Three typical masks were displayed, all by means of the box generator. In addition, one mask was also displayed by individual vectors. The times for performing various tasks by the display generator were measured. The results are shown in Table I. A comparison of the line drawing time for the two cases of Mask 3 (see table) shows that the box generator does indeed increase the mask drawing speed by a substantial factor, but the improvement is lost in the time used by other parts of the system. The character generation time is

disproportionately large because the character generator was deliberately slowed down to accommodate the storage scopes. The new character generator should improve character writing time by a factor of three or four. The other possibilities for improvement, display system overhead and CPU display servicing time, are being studied.

A surprising result is the small amount of time used in positioning the various display elements. It appears that the display data are structured in an orderly way so that the CRT deflection systems do not have to spend a great deal of time to arrive at the starting positions.

3. Character Generator

A new character generator based on the stroke writing principle and using D/A conversion techniques developed for the TX-2 graphics display has been built and is now being checked out. Stroke data are stored digitally in a read-only memory. During checkout, however, a small plug-board memory is being used to confirm the stroke data for the TX-2 character set, a portion of which is shown in Fig. 1. The read-only memory (790 words × 29 bits) will be purchased as soon as the character designs are firm.

The block diagram in Fig. 2 shows the essential parts of the generator. The data required for each stroke are given below.

		Number of Bits
X starting point	x_0	5
Y starting point	y_0	5
X staircase multiplier	x_1	5
Y staircase multiplier	y_1	5
Number of steps	n	5
Control		4

Before the start of each stroke the stroke data x_0, y_0, x_1, y_1, are read out of the read-only memory and stored in the input registers of the D/A converters (DAC's). N is stored in a control register. The staircase generator is then started. The staircase waveform serves as an accurately controlled approximation to a ramp. The staircase generator is a DAC whose digital source is a binary counter. The counter is clocked until N counts have occurred at which point the control inhibits further counting. N may have any value from 1 to 32. The staircase output for N = 5 is shown in Fig. 3. The staircase voltage is multiplied by x_1 and y_1 determining the slope of the stroke and then added to the starting points x_0 and y_0 to form the deflection waveforms. The staircase may be expressed in terms of the unit step as

$$S(t) = \sum_{n=1}^{N} U(t - nT)$$

$U(t - nT)$ is a step at $t = nT$

$$T = \frac{1}{\text{clock frequency}}$$

and the deflection waveforms as

$$\left.\begin{aligned} X &= x_0 + x_1 \left[\sum_{n=1}^{N} U(t - nT) \right] \\ Y &= y_0 + y_1 \left[\sum_{n=1}^{N} U(t - nT) \right] \end{aligned}\right\} \quad n = 1, 2, 3, \ldots, N$$

The equations of the smoothed equivalent waveforms are

$$X = x_0 + x_1 \left[\frac{t}{T} - \frac{1}{2} \right] \approx x_0 + \frac{x_1}{T} t$$

$$Y = y_0 + y_1 \left[\frac{t}{T} - \frac{1}{2} \right] \approx y_0 + \frac{y_1}{T} t$$

At the beginning of the next stroke the staircase counter is cleared simultaneously with the strobing of new stroke data into the DAC registers and counting proceeds for the next stroke. The delay in setting up new stroke data is short enough to allow the CRT beam to stay on between contiguous strokes. Noncontiguous strokes require blanking between strokes. The writing speed of the character generator, i.e., the clock rate of the staircase generator is program variable. A higher writing rate will be used for refresh scopes and a slower rate for storage scopes. For example, the letter A is drawn in $20\,\mu$sec for the refresh scopes and $70\,\mu$sec for the storage scopes.

II. GRAPHICS AND APPLICATIONS

A. Regional Planning Program

In conjunction with the Department of Regional Planning and Landscape Architecture of the University of Pennsylvania, a system has been built to aid in regional planning. The study has involved two areas of computer graphics:

    (1) The display of maps described by giving their point-by-point gray-scale values.

    (2) The human factors' engineering of the use of such displays.

One of the tasks of regional planners is the determination of land use. Specifically, given data about a given region, and a set of demands for land within the region, they are faced with the task of allocating land within the region for different uses, so as to maximize its usefulness to all interests concerned. Although the measures of usefulness are still in their early stages of development (and note that the complete ecosystem must be considered in these measures), there are certain aspects of the task which can be usefully automated. The study this system represents is aimed at determining what form of program best aids in the land use allocation task.

The system provides a planner with the ability to operate on maps. The region of land being considered is described in terms of a number of features; for example, slope, drainage characteristics, soil type, vegetation, etc. The region being considered is described by dividing it into squares and assigning one of eight values to each of these squares. The current system treats a square area with 100 unit squares to the side. Thus each area is described by 10,000 values, each chosen from a selection of eight possible values.

The system provides for creating these maps interactively using the dynamic scope (see Fig. 4). The planner outlines areas on the map and the value associated with the area, and the system fills in the values. This method of feature definition is quite inadequate, for it is very difficult to trace a map in any reasonable way. The best way would be to trace a map laid on top of the input tablet. However, these tablets have sufficient distortion to render the resulting map badly distorted. The alternative is to place the map to be traced over the dynamic scope face and then trace it using the tracking dot. However, the parallax caused by the distance between

the tube phosphor and the tube surface, and the curvature of the tube surface render this method impractical as well. Automatic digitization techniques are being used by the University of Pennsylvania in Philadelphia, and they have proved quite satisfactory. Hence little effort has been expended to improve the input properties of the Lincoln system. Instead routines to read map descriptions from magnetic tape have been provided.

Display of complete maps is done on a storage scope, as a dynamic display of 10,000 values has unbearable flicker.

A gray-scale display is achieved by displaying different alphabetic characters according to the value to be represented. This is similar to the technique used with a line printer. However, as positioning on the storage scope has 10-bit resolution, the characters can be displayed so that they abut one another. Appropriate choices of spacing and letters provide an adequate eight-level gray scale (Figs. 5 and 6).

Once features have been defined, or read in from tape, they may be displayed. However, the real purpose is to create new maps on the basis of the already known ones. Specifically, regional planners want to know where various combinations of values of features occur. Means of specifying such intersections of features are provided by the system. For each such intersection, a value (choice of eight) is specified. A new map is plotted with the associated value wherever the intersections specified occur. A default value is specified. This map may be built up one intersection at a time, or it may be done more quickly by specifying sets of intersections. Errors may be deleted. The map so produced can be given a name, and treated from then on as any of the original features. Thus new maps can be made on the basis of it. The current system restricts intersections to be made on the basis of at most four other maps. However, by cascading maps an arbitrary number of features can be reflected in a final map.

Hard copy of many things in the system are obtainable: maps, intersection specifications, counts of intersection areas. The Xerox line printer on TX-2 is used and the gray-scale maps are not very satisfactory.

Experience with the system has been limited. However, it appears that it provides such a significant speedup of the map-making process that it is best characterized by a change in kind in the capability of the planners in land-use allocation studies.

B. Semiconductor Mask Design Program

1. Introduction

For the past two years a program to aid designers of large-scale integrated circuits has been under development. It has been an important and challenging application for the graphics and programming language support systems on TX-2. During this period four versions of this large and complicated program have been generated, a fact which attests to the ease with which such a program may be written and modified using the available supporting facilities. Each version has been more useful to the circuit designers than the previous one, but each has also been more demanding in terms of system resources. In its current version, the mask program is a helpful tool in the iterative process of designing integrated circuit layouts, a convenient tool for editing layouts when errors are detected, and a vital tool for automated artwork generation.

The main problem with the program as it has evolved is its large and inefficient use of system resources with consequent high cost and low availability. The question of the efficiency

of large interactive programs with or without time sharing has been discussed elsewhere† and will not be discussed here. Instead, this report will review the evolution of the mask design program, describe the requirements of the circuit designer, and discuss the effectiveness of the current version of the program in terms of the design problem. Finally, a new and radically different approach to the graphical problems of circuit layout and mask generation will be described.

2.   Evolution of the Mask Program

Version I: Masks were produced by combining component definition procedures, component instance procedures, and a utility package for displaying and punching at program compile time. No provision was made for modifying the masks interactively.

Circuits produced:

(a)   Fuse test chip
(b)   Thermal test chip
(c)   Read-only-memory (ROM) circuit chip (see Fig. 7)
(d)   Process evaluation chip
(e)   Collector-diffused-isolation (CDI) 3-input ECL gate.

These chips were either simple or very iterative.

Version II: This program allowed masks to be designed interactively, i.e., component instances could be added, edited and viewed by means of tablet commands. Component definitions still had to be entered at compile time. This program was used to produce a simple 3-input ECL AND gate (see Fig. 8).

Version III: In this version, the command strategy was changed. Instead of having a tablet command cause everything "near" the tablet stylus to be affected, this version had two stroke commands: first, the elements to be affected were "selected," then the command was given. Another major improvement was the inclusion of groups — large numbers of instances of components treated as a single component. Other improvements were: an indication on the scope face of the absolute coordinates of the component (this was difficult to know because of windowing), a capability to step and repeat components, and finally, provision for typewriter input as well as tablet input.

---

† W. R. Sutherland, J. W. Forgie and M. V. Morello, AFIPS Conference Proceedings (AFIPS Press, Montvale, N. J., 1969), Vol. 34, pp. 629-636. Graphics, Semiannual Technical Summary to the Advanced Research Projects Agency, Lincoln Laboratory, M.I.T. (31 May 1968), p. 6, DDC AD-671125.

Circuits produced:

(a) An SMX-14 gate chain, 100×100 mils, which was reasonably iterative.

(b) A BB chip – 80 identical ECL gates, 120×130 mils. As an example of the complexity here, there were 30,000 boxes on one mask of the BB chip, with perhaps 100,000 boxes in total (Fig. 9)

(c) A 4-bit adder, which was effectively a different metallization pattern on the BB chip.

(d) A CDI gate chain similar to the SMX-14.

Version IV: This is the version currently in use. The major improvements of this version over the previous one is the arrangement of the LEAP structure to include more efficient scaling and windowing at the expense of selection time. Second, component definitions are now part of the data base and not part of the program. Third, the capability exists to output a text file description of the data base which includes relationships between boxes.

Component definitions produced:

Two bipolar geometries; the CDI process, and an MOS process.

Circuits produced:

A 2-bit ECL array multiplier element has been designed, which is 40×60 mils and not iterative at all (see Fig. 10).

The multiplier would have been essentially impossible to build with Version III, because of the time necessary to window and scale.

3. Mask Design Problem

A mask designer proceeds through several steps from the original conception of a circuit to its final realization:

(a) A rough block structure of the circuit is produced.
(b) A detailed circuit diagram at the level of transistors and resistors is produced.
(c) A paper and pencil sketch of the chip layout is produced and turned over to a skilled draftsman.
(d) The draftsman produces high quality artwork of the masks.
(e) The designer and draftsman carefully check that the artwork accurately reflects the original circuit and the geometry rules of the fabricator, repeating steps (d) and (e) until all constraints are satisfied.

Since the ability to feedback from one step to another is crucial, changes in the layout should be easy to make at any step. In the past, this has been an expensive and complicated

procedure. The current mask program solves this problem. It is essentially an editor of art-work: a not too intelligent but highly responsive tool.

At present, the checking involved in step (e) is carried out without effective machine aids Designers use hard copy from an incremental plotter in this step for several reasons: the reso-lution of the display is not adequate, the flicker is too great, the designers feel they need con-tinuous windowing, and the computer is not available for the many hours required. The computer, of course, is capable of doing other things to aid this process. For example, it could compare the mask with the circuit diagram, if it were given information about the correspondence between components. This syntax checking would remove several hours from step (e) of the design process. It should be noted that many programs already exist for the semantic checking of cir-cuit diagrams, such as ECAP, and a complete system would include such programs.

Masks for the complicated chips currently being designed tax the capacity of the system. To extend operations to the still more complicated chips expected in the near future would re-quire major effort to improve display flicker, response time, and data handling capability, not to mention the desirability of providing new aids in the checking phase of the design process. While it may be feasible to achieve the necessary improvements within the existing mask pro-gram framework, the prospects are not attractive, and no further development of the program is contemplated. Instead, a radically different, high performance, new system for integrated circuit layout and mask making is being designed for TX-2.

4. A New Circuit-Layout and Mask-Making System

The new system's basic characteristics are (a) the use of an X-Y raster display and (b) the splitting of the mask program into two distinct parts: a layout part, and a mask making part.

The X-Y raster display technique, in which the beam first traces a horizontal raster pattern and then a vertical raster pattern, and in which the elementary displayed unit is a short (hor-izontal or vertical) "stroke," is capable of providing considerably better resolution than either the vector drawing technique (presently used in TX-2) or the ordinary X-raster technique. It can provide better resolution than the vector technique because the beam is moving continuously at a constant rate, and better resolution than the X-raster technique because vertical lines are displayed as lines, not as accumulations of points. It cannot directly display diagonal lines, a limitation insignificant for mask making of integrated circuits since diagonal lines can be dis-played quite adequately as staircase patterns.

The splitting of the mask program into two parts recognizes that, during the layout of the circuit, information should be displayed not in its final mask form, but instead in the best form suitable for layout purposes. For example, a transistor should be displayed by its emitter, col-lector, and base contacts only, without any other details being shown. Components can then be rearranged, deleted, inserted, etc., with the information presented at all times in the best visual form. Once the layout is completed, the second part of the program takes over, convert-ing the "best visual form" information into the multitude of the actual masks required.

The display will provide an 8 × 11-inch working area. In it, a 0.1-inch grid will be dis-played, with, say, every fifth and tenth lines accented. Global coordinate numbers will be shown on the grid. Superimposed on this grid, three "working sheets" will be displayed, rep-resenting three separate "levels" of the circuit (such as (a) transistor emitter, base, and col-lector contacts, (b) first level metal, (c) second level metal). The overall visual impression will be as if three transparent drawing sheets are used on top of a 0.1-inch grid paper. Lines

can be drawn on these sheets either on the 0.1-grid lines or half-way between grid lines. Thus
the elemental stroke is 0.050 inch long. Two hundred and twenty such strokes are contained in
each horizontal raster line, and 160 in each vertical raster line. The intensity of each stroke
is a weighted average of the intensity of the "grid sheet" plus the three working sheets at this
stroke location. The intensity weight of each sheet will be adjustable by front-panel controls so
that any particular sheet or sheets can be made more (or less) prominent.

Each stroke will be stored as 1 bit/sheet, i.e., 4 bits for the 4 sheets used. The sheets dis-
played are actually parts of large sheets containing the entire circuit. The bits corresponding
to these large sheets will be stored on the Fastrand drum, while the portion actually displayed
will be duplicated in core. Approximately 16,000 32-bit words are required in core for the
8×11-inch display. Total drum storage requirements depend on the size and complexity of the
integrated circuit being designed; they are about 300,000 words for a moderately complex cir-
cuit containing 20 gates. The X-Y raster display will obtain the information to be displayed by
reading the 16,000 words continuously. Reading repetition rate will be 25/second so that the dis-
play will be flicker free.

The window displayed will be capable of being moved continuously, in movie fashion, by
transferring new information from the drum to the core; continuous motions of 5 inches/second
will be possible without any lag. The system will enable exact drawing of complicated contours
(say, for metallization levels) in no more time than that required for freehand drawing on grid
paper. Erasing, correcting, and moving should be faster than the corresponding operations on
paper. In general, speed and convenience of usage will be stressed during the design of the
system.

Response with the new system should be very rapid for the frequent commands involving
viewing and drawing. More dedicated core memory will be required than is the case with the
current mask program, but overall core requirements should be substantially less for most of
the time that the designer or draftsman is using his console. The system should therefore be
compatible with other time-shared use of TX-2, allowing use of the system throughout the work-
ing day. This greater availability combined with the high resolution and flicker-free character-
istics of the X-Y raster display should eliminate the need for hard copy in the checking phase of
the design process.

C. Graph Theory

1. High-Speed Planarity Testing

A high-speed algorithm for testing the planarity of a graph appears crucial to the develop-
ment of effective computer aids for circuit layout, flowcharting, and other problems involving
the representation of complex networks in graphical form. The term "planarity testing" is used
here to refer collectively to the testing of graphs for planarity, extraction of planar subgraphs
from nonplanar graphs, and layout of planar graphs. Algorithms for planarity testing exist in
relative abundance, but with varying degrees of practicality;[†] the object of current research is
to produce a new algorithm with greatly reduced time requirements as well as reduced storage
requirements.

[†] R. W. Shirey, "Implementation and Analysis of Efficient Graph Planarity Testing Algorithms,"
Ph.D. thesis, University of Wisconsin (1969).

Considerable progress has been made in the past few months toward completing the development and validation of such a planarity-testing algorithm. A 160-page report[†] has been written that describes and proves the sufficiency of the new graph-theoretic methods that have been developed. Prospects look encouraging for the successful completion of the algorithm.

Informal estimates indicate that computing time will be reduced by a factor of n (where n = number of nodes in the graph), in comparison with time requirements of earlier published methods of planarity testing. As of now, the best published estimate for computing time has been proportional to n^3 (see footnote on p. 12). Estimates based on the new algorithm suggest a time proportional to n^2; storage requirements will definitely be reduced as well. Thus, for a circuit with n = 100 nodes, the improvement in speed is expected to be two orders of magnitude.

2.   Proposed Algorithm

The algorithm is motivated most easily by example. Consider the graph G specified geometrically below:
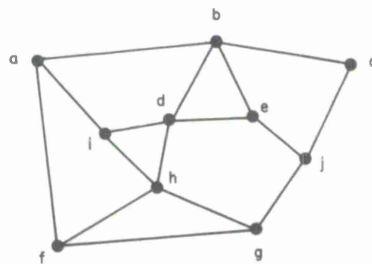
Observe that the sequential ordering of vertices

$$O = \langle c, j, e, g, f, a, b, h, d, i \rangle = O_1, O_2, \ldots, O_n$$

has the unusual property that a planar realization of G can be constructed as follows:
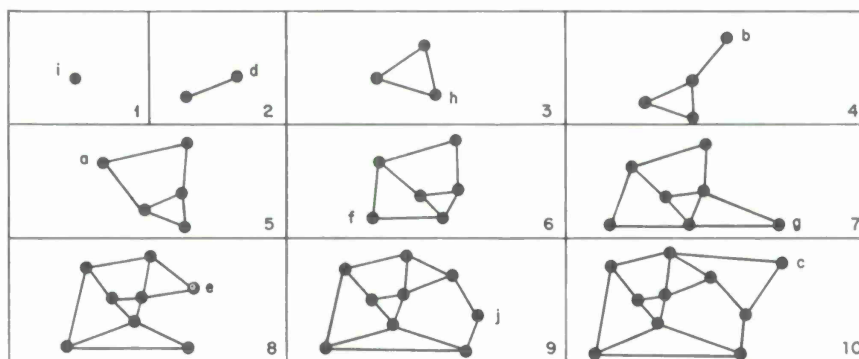
The idea of this construction is to generate a sequence of planar realizations $S_n^* \subset S_{n-1}^* \subset S_1^* = G^*$ of subgraphs of G (where $S_1^* = G^*$ is a planar realization of G) as follows:

_____

† L. F. Mondshein, private communication.

(a) First, let S_n^* = the last node ($O_n = i$) of the sequence.

(b) Given S_k^*, construct S_{k-1}^* as follows:

place node O_{k-1} <u>outside the periphery</u> of the graph S_k^*, and add (in a planar fashion) arcs corresponding to all edges between O_{k-1} and the vertices in S_k^*.

A fundamental feature is that, when constructing S_{k-1}^*, no modification of S_k^* — i.e., "no backtracking" — is permitted.

Two major problems confront any attempt to transform the above idea into a rigorous planarity-testing algorithm. First, and most obvious, comes the problem of generating the vertex sequence O to be utilized in the algorithm. The structure of this sequence must somehow guarantee that the algorithm will operate properly, where by "properly" we mean: If G is a planar graph, then the algorithm will yield a planar realization of G. The input to a planarity-testing algorithm consists solely of the sets of vertices and edges of the graph, and this abstract specification includes no obvious clues to the structure of a suitable sequence O.

A second problem stems from the condition that no backtracking is to be allowed. An example will help elucidate this problem. Let G be the graph specified geometrically below.
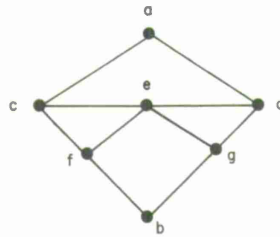
Let O be the sequence $\langle b, a, g, d, f, e, c \rangle$. Now observe the results of applying the procedure described above, using this sequence:
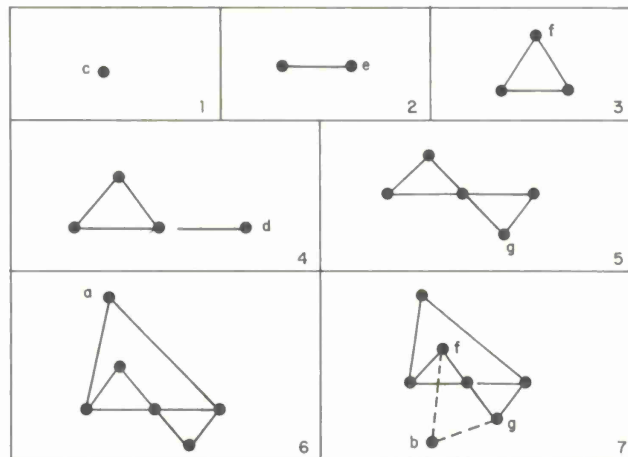
Note that, even though G is planar, the final step of the algorithm is blocked: regardless of where b is placed outside the periphery, there is no way of connecting b and f in a planar fashion.

One might ask: Why not forego the rule forbidding backtracking, and modify the placement of node f in the obvious way? The answer is that the efficiency of the proposed algorithm rests heavily on the absence of backtracking. If allowed, backtracking could lead to a very costly amount of recursive searching and modification.

One might also ask: Why insist on placing successive nodes outside the periphery; why not allow arbitrary placement? The answer is that this restriction, if workable, will reduce the amount of graph-structure to be considered at each step, and therefore, the amount of computation (namely, we need consider only the periphery); in fact, permitting placement within the periphery leads to methods that are essentially equivalent to earlier (and slower) algorithms.

Current research has produced substantial steps toward overcoming the problems just discussed. An algorithm based on the first example above has been rigorously specified. The input to this algorithm consists of the set of vertices and edges of an abstract graph, together with a vertex sequence O. As is apparent, not just any vertex sequence can be permitted. A fundamental accomplishment has been the description of a relatively uncomplicated, graph-theoretic condition on such sequences — called the "anchored vertex" condition — that is sufficient to guarantee that the algorithm works properly (in particular, backtracking is never required). Specifically, a theorem has been proved which states: given any graph G that is 3-connected (a readily justified assumption) and any "anchored vertex sequence," the algorithm (which is free of backtracking) produces a planar layout if and only if G is planar.

Thus the problem has been reduced to the construction of an anchored vertex sequence, and the major portion of research has been directed toward this issue. In particular, a new constructive theorem concerning disjoint paths in 3-connected graphs has been proved. Based on this theorem, an iterative technique has been developed for use in attempting to generate an anchored sequence on a 3-connected graph. It has been proved that the problem of generating an anchored sequence reduces to the much simpler problem of accomplishing the initial step of each iteration.

Most recently, the problem has been simplified still further. At present, the iterative technique just mentioned will generate an anchored sequence except in special cases. Work is continuing toward augmenting the technique to handle these cases.
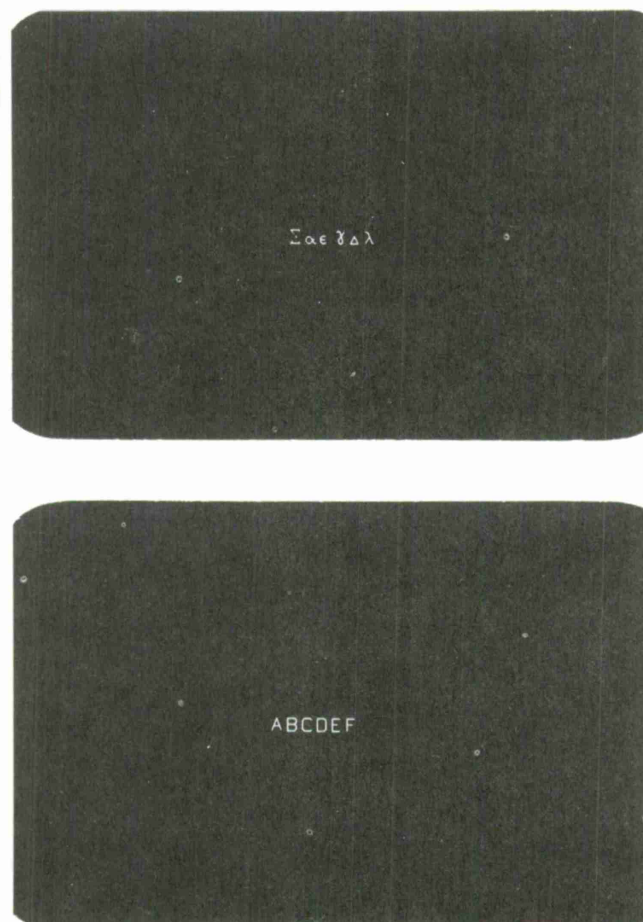
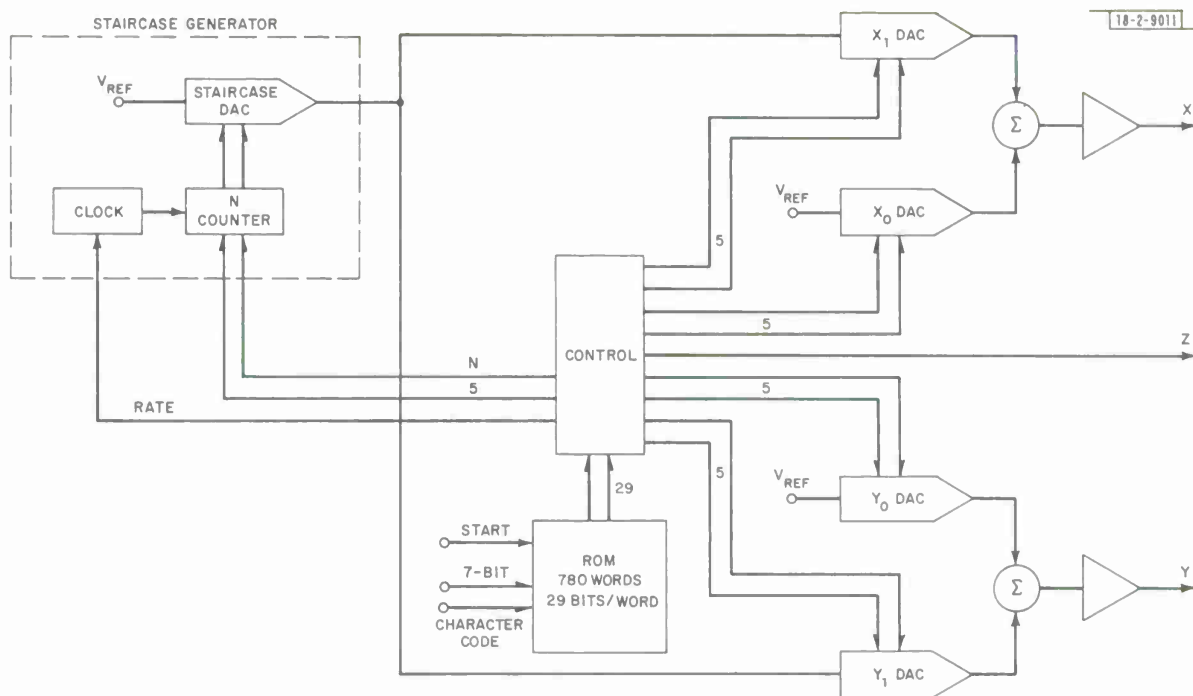Fig. 1.   A portion of the TX-2 character set generated by the new character generator.

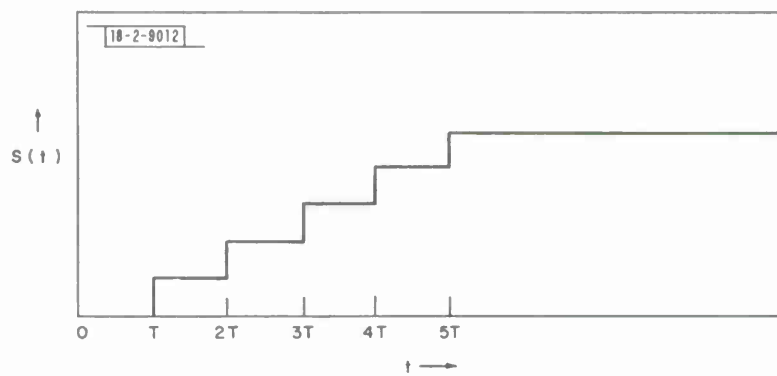Fig. 2. TX-2 character generator.

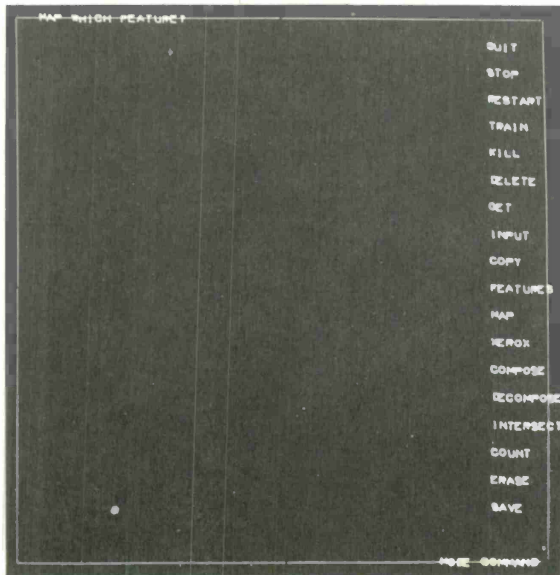Fig. 3. Staircase voltage for character generator.

Fig. 4. Command mode display (on the dynamic scope). Each light button causes entrance to a different subsystem of the program.
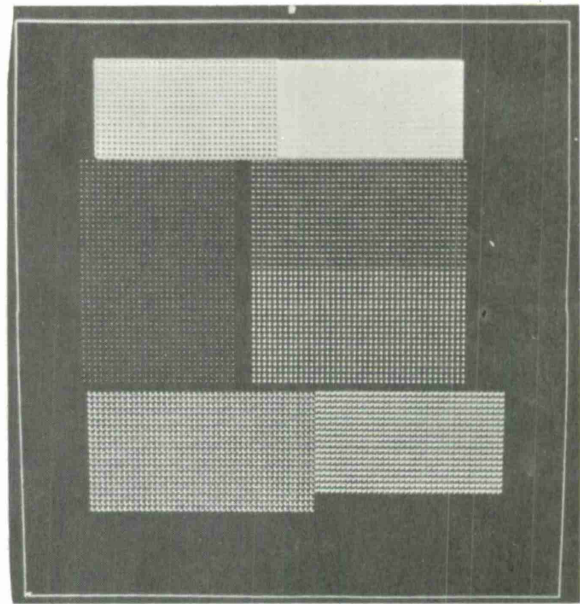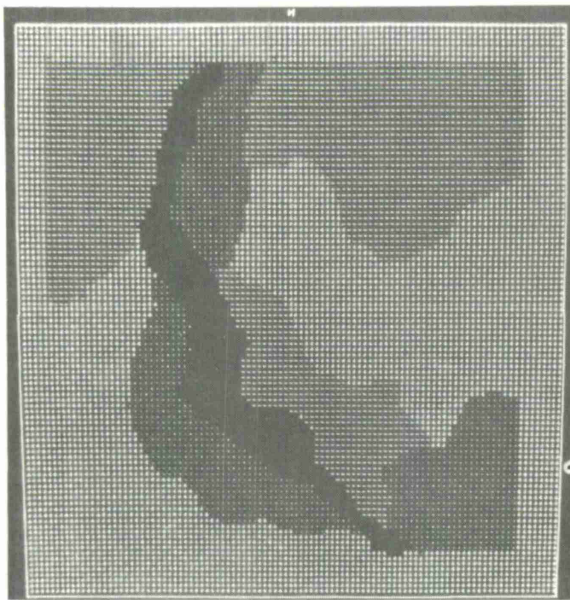
Fig. 5. Storage scope displays. (a) Maps feature named H with only four levels of grey; (b) maps atypical feature B designed to demonstrate eight levels of grey.

Fig. 6. Intersect mode display. (a) On dynamic scope an intersection is being performed among features named OWNER, ELEVATION, SOILS, and B. Each line of the intersection specifies areas of the map by values of the features being intersected, and values to be associated with each of these areas. The default value is assigned to all areas unspecified in any other fashion. (b) Example of map created by intersecting other features. It is specified as a PARTIAL INTER-SECTION as more lines of intersection may be added to the definition.
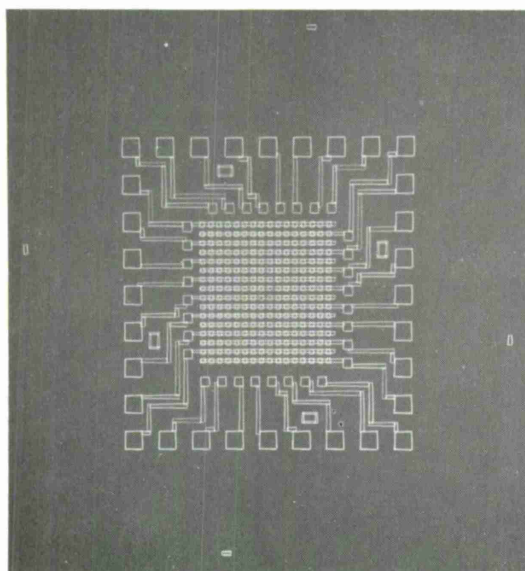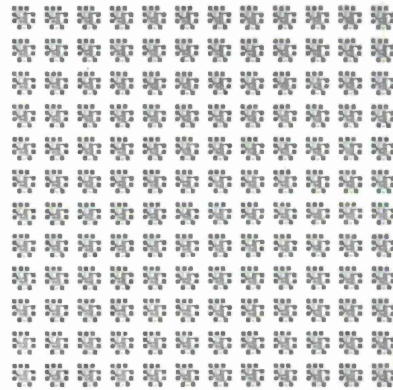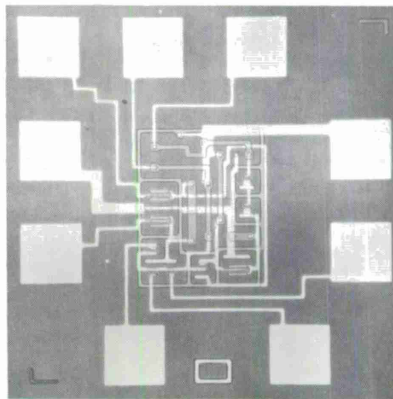
(a)
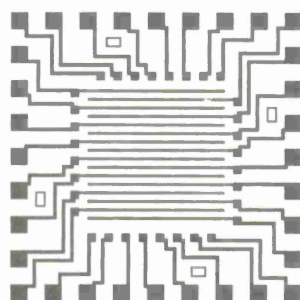
(b)
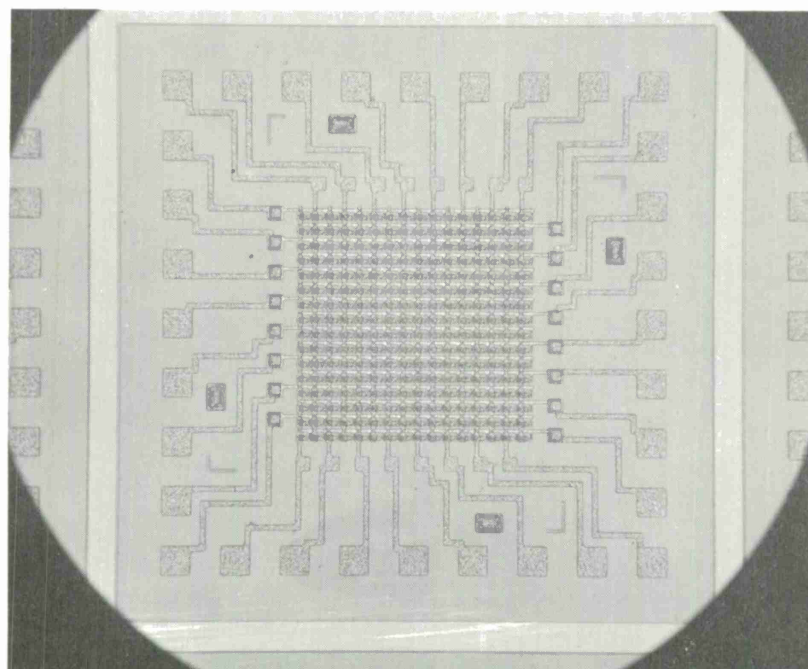
(c)

Fig. 7.   ROM chip.   Implemented with Version I, this highly iterative read-only memory chip measures 66×66 mils.

(a)

(b)

(c)

Fig. 8.   3-input ECL AND gate.   Implemented with Version II, this single
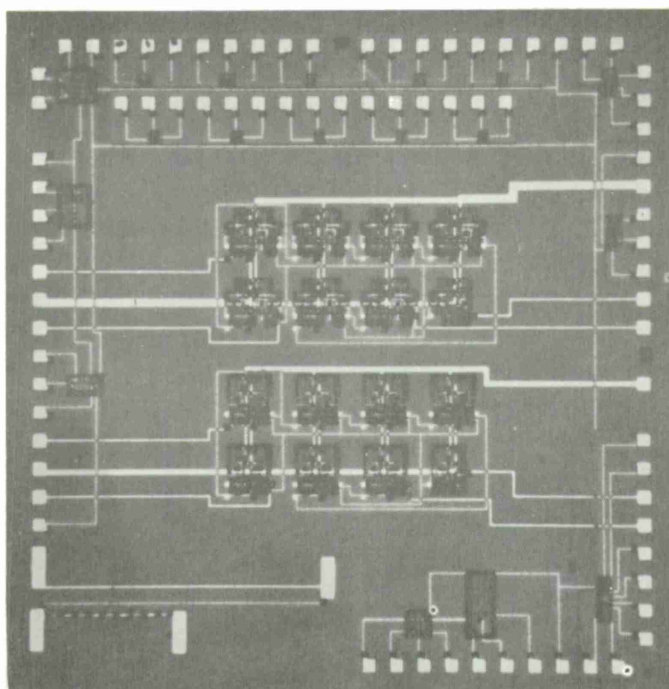3-input ECL AND gate measures 25×25 mills.

Fig. 9. BB gate chain. Implemented with Version III, this gate circuit contains 80 identical gates with moderately iterative metallization, and measures 120 × 130 mils.
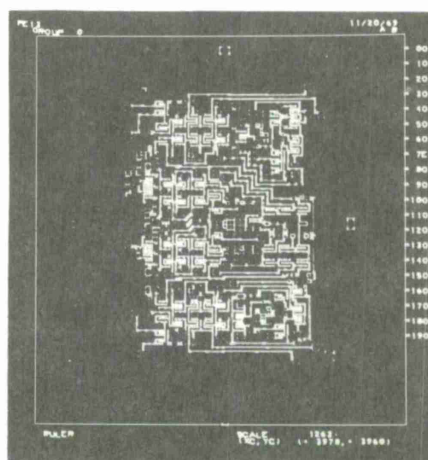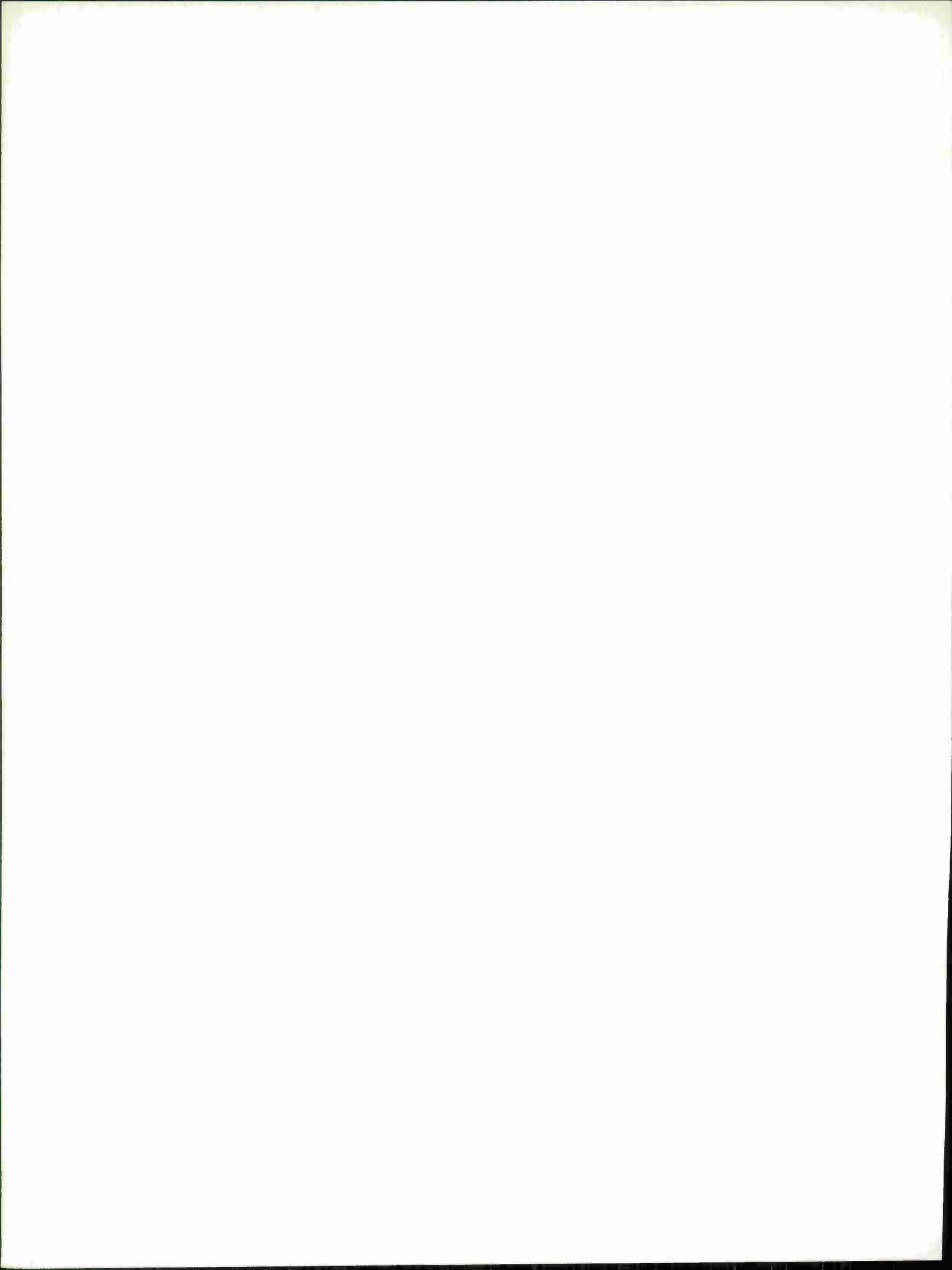
Fig. 10. ECL array multiplier element. Designed but not implemented, using Version IV, this multiplier element has no iterative structure at all and measures 40 × 60 mils.

DOCUMENT CONTROL DATA – R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*	2a. REPORT SECURITY CLASSIFICATION
Lincoln Laboratory, M.I.T.	Unclassified
	2b. GROUP None

3. REPORT TITLE

Graphics, Semiannual Technical Summary Report to the Advanced Research Projects Agency

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Semiannual Technical Summary (1 June through 30 November 1969)

5. AUTHOR(S) *(Last name, first name, initial)*

Forgie, James W.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
30 November 1969	32	4

8a. CONTRACT OR GRANT NO. AF 19 (628)-5167	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. ARPA Order 691	Semiannual Technical Summary for 30 November 1969
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	ESD-TR-69-384

10. AVAILABILITY/LIMITATION NOTICES

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency, Department of Defense

13. ABSTRACT

Graphical output and interactive input routines have been written for BCPL, offering a useful alternative to assembly language for writing graphical subsystems on TX-2. Design work is under way on the system architecture and software for a new terminal support system intended to serve as many as 20 interactive but nondynamic graphic consoles. A storage scope editor has been implemented on TX-2 with the intent of exploring some of the problems to be encountered in the new system which will have storage scopes for display output.

Experiments with the color display on TX-2 await the delivery of a new CRT with longer persistence. A box, or rectangle, generator was designed and installed in an attempt to reduce the display flicker for the semiconductor mask design application. The resulting improvement prompted detailed measurement of display system performance. A new character generator based on the stroke writing principle has been built and is being checked out.

A program written to demonstrate the application of interactive graphics to regional planning has incidentally shown that the storage scope can provide quite adequate eight-level gray-scale area maps. Conclusions drawn from two years' experience with programs that aid in the design of semiconductor masks have led to the design of a radically different, high performance, new system for integrated circuit layout and mask making. Progress has been made in the development and validation of a high-speed algorithm for testing the planarity of a graph which is expected to have application in the layout problem.

14. KEY WORDS

graphical communication	time sharing	display systems
TX-2	man-machine	programming languages